

A PRELIMINARY REPORT ON

SUGGESTING RELEVANT QUESTIONS FOR
A QUERY USING NATURAL LANGUAGE
PROCESSING TECHNIQUES.

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE
IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF ENGINEERING
(COMPUTER ENGINEERING)

SUBMITTED BY

HRUSHABH HIRUDKAR
ANUJ KANETKAR
SHRINIWAS NAYAK

Exam No : 71700888G
Exam No : 71700929H
Exam No : 71701044K



DEPARTMENT OF COMPUTER ENGINEERING

PUNE INSTITUTE OF COMPUTER TECHNOLOGY

DHANKAWADI, PUNE - 411043

SAVITRIBAI PHULE PUNE UNIVERSITY

2019-2020



CERTIFICATE

This is to certify that the project entitled

SUGGESTING RELEVANT QUESTIONS FOR A QUERY USING NATURAL LANGUAGE PROCESSING TECHNIQUES

Submitted by

HRUSHABH HIRUDKAR
ANUJ KANETKAR
SHRINIWAS NAYAK

Exam No : 71700888G
Exam No : 71700929H
Exam No : 71701044K

are bonafide students of this institute and the work has been carried out by them under the supervision of **Dr. A. S. Ghotkar**, and it is approved for the partial fulfillment of the requirement of Savitribai Phule Pune University, for the award of the degree of **Bachelor of Engineering** (Computer Engineering).

Dr. A. S. Ghotkar
Guide,
Dept. of Computer Engg.

Mrs. M. S. Takalikar
Head,
Dept. of Computer Engg.

Dr. P. T. Kulkarni
Principal,
Pune Institute of Computer Technology

Place: Pune

Date:

ACKNOWLEDGEMENTS

It gives us great pleasure in presenting the preliminary project report on ‘**Suggesting relevant questions for a query using natural language processing techniques**’.

We would like to take this opportunity to thank our internal guide **Dr. A. S. Ghotkar** for giving us all the help and guidance we needed. We are really grateful for her kind support.

We are deeply grateful to **Dr. S. S. Sonawane** for her valuable suggestions without which it would not have been possible for so to implement the system to its current form

We are also grateful to **Prof. Mrs. M. S. Takalikar**, Head of Computer Engineering Department, PICT for her indispensable support, suggestions.

In the end our special thanks to **Ms Jugnu Manhas** and **Mr Nikhil Malhotra** for providing valuable insights for our Project. It was our pleasure to work in the Maker’s Lab, Tech Mahindra.

We would also like to thank our family and friends, without their support this would not have been possible.

Hrushabh Hirudkar
Anuj Kanetkar
Shriniwas Nayak
(B.E. Computer Engg.)

ABSTRACT

Suggesting similar questions for a user query has many applications ranging from reducing search time of users on e-commerce websites, training of employees to holistic learning for students. Mainly two approaches are studied for finding the similarity namely syntactic and semantic. In this article, a combined approach is proposed for determining textual similarity that introduces a robust weighted syntactic and semantic similarity index for determining similar questions from a predetermined database. Comprehensive set of experiments have been carried out to justify the efficacy of the proposed approach over the existing literature.

Keywords:

NLP, Information Retrieval, Lexical Similarity, Syntactic Similarity, Semantic Similarity, NLU

Contents

List of Abbreviations	i
List of Figures	iii
List of Tables	iv
1 Introduction	1
1.1 Overview	2
1.2 Motivation	2
1.3 Problem Definition and Objectives	3
1.4 Project Scope and Limitations	3
1.5 Methodologies of Problem Solving	4
2 Literature Survey	5
3 Software Requirements Specification	8
3.1 Assumptions and Dependencies	9
3.2 Functional Requirements	9
3.2.1 Suggestion of questions	9
3.2.2 Collection of feedback	9
3.2.3 Generating Reports	10

3.3	External Interface Requirements	11
3.3.1	User Interfaces	11
3.3.2	Communication Interfaces	12
3.4	Nonfunctional Requirements	12
3.4.1	Performance Requirements	12
3.4.2	Safety Requirements	12
3.4.3	Security Requirements	13
3.4.4	Software Quality Attributes	13
3.5	System Requirements	13
3.6	Analysis Models: SDLC Model	14
4	System Design	16
4.1	System Architecture	17
4.1.1	Overall Hierarchy	18
4.1.2	Syntactic similarity architecture	19
4.1.3	Semantic Similarity Architecture	20
4.1.4	Database Tagging Architecture	21
4.1.5	Self Learning Lambda Architecture	22
4.2	Data Flow Diagrams	23
4.2.1	DFD Level 0	23
4.2.2	DFD Level 1	24
4.3	UML Diagrams	25
4.3.1	Use Case Diagram	25
5	Project Plan	26
5.1	Project Estimate	27

5.1.1	Reconciled Estimates	27
5.1.2	Project Resources	27
5.2	Risk Management	27
5.2.1	Risk Identification	28
5.2.2	Risk Analysis	28
5.2.3	Overview of risk mitigation, monitoring and management . .	28
5.3	Project Schedule	29
5.3.1	Project Task Set	29
5.3.2	Task Network	29
5.3.3	Timeline Chart	30
5.4	Team Organization	31
5.4.1	Team Structure	31
5.4.2	Management reporting and communication	31
6	Project Implementation	32
6.1	Overview of Project Implementation	33
6.2	Tools and Technology Used	34
6.3	Algorithms Details	35
6.3.1	Algorithm for Syntactic Analysis	35
6.3.2	Algorithm for Semantic Analysis	36
6.3.3	Algorithm for Database tagging	37
6.3.4	Algorithm for Self Learning Lambda	37
7	Software Testing	39
7.1	Types Of Testing	40
7.1.1	Functional Testing	40

7.1.2	Non-Functional Testing	40
7.2	Test Cases And Test Results	41
7.2.1	Unit Testing	41
7.2.2	Integration Testing	42
7.2.3	Stress Testing	43
8	Results	44
8.1	Outcomes	45
8.2	Screen Shots	46
9	Conclusion And Future Scope	53
9.1	Conclusion	54
9.2	Future Scope	54
9.3	Application	55
Appendix A	Feasibility Study	56
A.1	Problem Statement Feasibility	57
Appendix B	Details of Papers	59
Appendix C	Plagiarism Report	60
10	References	61

List of Abbreviations

ABBREVIATION	ILLUSTRATION
NLP	Natural Language Processing
NLU	Natural Language Understanding
SDLC	Software Development Life Cycle
NLTK	Natural Language Tool Kit
UI	User Interface
UML	Unified Modeling Language
GCP	Google Cloud Platform
GPU	Graphics Processing Unit
JSON	JavaScript Object Notation
P	Polynomial Time Problem
NPC	Non deterministic Polynomial Time Complete Problem
NPH	Non deterministic Polynomial Time Hard Problem
SSRD	Sum of Squared Rank Distance
NFR	Non Functional Requirements
DB	Database
DFD	Data Flow Diagram
TC	Time Complexity

List of Figures

3.1	Agile Model	14
4.1	Bird's Eye View - Overview	18
4.2	Syntactic Similarity - Cosine Similarity	19
4.3	Doc2Vec Architecture	20
4.4	Database Tagging Architecture	21
4.5	Self Learning Lambda Architecture	22
4.6	DFD Level 0	23
4.7	DFD Level 1	24
4.8	Use Case Diagram	25
5.1	Task Network	30
5.2	Timeline Chart	30
7.1	Unit Testing	41
7.2	Integration Testing	42
7.3	Stress Testing	43
8.1	JSON Output	46
8.2	User Interface - File Selection	47
8.3	User Interface - Answers	48
8.4	SSRD vs Lambda Graph	49

8.5	Observation File	50
8.6	Log File	51
8.7	Similarity Graph	52

List of Tables

2.1 Literature Survey	7
---------------------------------	---

CHAPTER 1

INTRODUCTION

1.1 Overview

In today's world where Machine Learning is ubiquitous, NLP as a sub domain has been able to prove its importance and wide range of applicability. NLP helps humans to interact with machines and receive responses from the machine that create an illusion that a human is answering. This kind of ease of access has never been realized before.

According to stack exchange the parent site of stack overflow boasts of 10 million views daily. All these views are for catering to their immediate requirement which stack overflow is available to fulfill but fails to develop the overall concept of the user.

Many technology enthusiasts including professionals working in the industry, researchers and students from all over the world use stack overflow as a ready reckon er for almost any difficulty they face in the technical field, while this proves extremely beneficial at the moment but does not help to develop the concept as a whole. Use of stack overflow solves the problem temporarily but does not help in enhancing knowledge, which in the long run, a layman may expect.

1.2 Motivation

In today's technologically advancing world, getting answers for questions, general or specific is very important to the development of the user and as a result, development of the overall community eventually. Thus, we propose to build a system to find the relevant questions to a query entered by the user to solve doubts thereby enhancing knowledge of the user. One of the most popular sources for people to find answers to their questions in StackOverflow. On StackOverflow people can easily find answers to the questions they have, but there is no suggestion for more questions on StackOverflow. We only get the answers to the question which we have asked, but if more similar questions are suggested this would generate inquisitiveness in the user regarding the subject.

1.3 Problem Definition and Objectives

We aim to solve this problem by generating similar questions to the query entered by the user. The generation of questions involves finding relevant questions from the existing database as well as generating new questions which are not present in the database. The objective is to give user the relevant questions he/she has entered or submitted to the system.

1.4 Project Scope and Limitations

This project uses a stackoverflow dataset but the system is able to produce desired output when fed with a similar dataset consisting of set of questions from any field. The project not only understands the fed query and presents syntactically similar questions from the dataset but also presents semantically similar questions that will help the user to understand the subject matter at hand better.

The project scope pans mainly over presenting questions based on an input user query when presented with a database. The project however does not extend to answering these questions or introducing completely new questions.

Following are the limitations of the project:

- Rich dataset having many diverse fields is difficult to construct
- It is difficult to create the same system for indic languages
- The system cannot handle grammatical errors present in the queries
- A subject matter expert needs to rank questions for self learning lambda system

1.5 Methodologies of Problem Solving

Choosing the right problem solving methodology is a very crucial part in the project, multiple brain storming sessions were carried out to finalize on the approach for solving the problem at hand. The basic technique is sub division of tasks and building a solution in bottom up manner. The different approaches that were implemented and compared in the survey are as follows :

- Syntactic Similarity Approach - This approach tries to suggest similar questions to the input query depending on the cosine similarity index.
- Semantic Similarity Approach - Under this approach similar questions are suggested using semantic similarity score which is calculated using dov2vec model.
- Combined Approach - This is problem solving methodology tries to combine the benefits of the above mentioned approaches and suggests the similar question by calculating a combined similarity index.

CHAPTER 2

LITERATURE SURVEY

The research paper[1] **A Survey of Text Similarity Approaches-International Journal of Computer Applications** discusses various approaches which can be used for textual similarity. It discusses the approaches which deal with syntactical similarity such as Cosine Similarity and n-grams and also discusses semantic similarity such as Knowledge Based Similarity.

Syntactical Similarity was studied further in the research paper[2] **Computing text similarity using Tree Edit Distance**. It discusses the n-grams syntactical similarity. It computes the similarity between two sentences using Tree Edit Distance.

Semantic Similarity was studied was studied in two parts. Firstly, Corpus Based and Knowledge Based Similarity was studied in the paper[3] **Corpus-based and Knowledge-based Measures of Text Semantic Similarity** and paper[4] **Semantic text similarity using corpus-based word similarity and string similarity**. It discusses the how Corpus Based and Knowledge Based Similarity can be used for Semantic Similarity. Secondly, The paper[5] **Unsupervised Sparse Vector Densification for Short Text Similarity** helps in understanding of semantic analysis. It uses the Explicit semantic analysis for the semantic similarity.

We also studied a new and better way for Semantic Similarity through the paper[6] **Robust semantic text similarity using LSA, machine learning, and linguistic resources**. It uses LSA with WordNet knowledge set to perform Semantic Similarity. This provides a better and efficient way for Semantic Similarity.

The research paper[7] **Siamese Recurrent Architectures for Learning Sentence Similarity** deals with the LSTM model for the Semantic Similarity of text. It used Siamese Network which helped to improve the efficiency of the model. This model can also be used for Text Generation.

From the research paper[8] **Isemantica: A command for text similarity based on latent semantic analysis** we learnt Latent semantic analysis uses truncated singular value decomposition to derive the hidden semantic relationships between words and texts. Isemantica provides a simple command for latent semantic analysis as well as complementary commands for text similarity comparison.

The paper[9] **Latent semantic analysis for application in a question answer system** uses a method that makes use of semantic vectors not only of individual nodes (concepts) in a graph representation of a document or corpus, but in using average of vector values of neighbors of node of interest.

Suggesting Relevant Questions For A Query Using Natural Language Processing Techniques

TITLE	DESCRIPTION	DATASET USED	RESULT
A Survey of Text Similarity Approaches- International Journal of Computer Applications (0975 – 8887) Volume 68– No.13, April 2013 [1]	Text Similarity Approaches		Study about various Text Similarity Approaches such as Character Based, Corpus Based, Knowledge Based
Computing text similarity using Tree Edit Distance - Sidorov et al (2015) [2]	TED for similarity between n-grams	Simple English Sentences	Gives better accuracy as compared to Levenhstein's SED
Corpus-based and Knowledge-based Measures of Text Semantic Similarity[3]	Approaches for measuring semantic similarity.		Detailed Study about Corpus based and Knowledge Similarity.
Unsupervised Sparse Vector Densification for Short Text Similarity - Song et al (2015)[4]	Combining ESA and word2vec for better similarity measure for shorter text	20 news group dataset for short text similarity	Combining yielded better result than individual operation
Robust semantic text similarity using LSA, machine learning, and linguistic resources - Kashyap et al (2015)[5]	Term alignment algorithm implementation	Simple paragraphs of literature essays	Best system for sentence-phrase, phrase-word similarity
Siamese Recurrent Architectures for Learning Sentence Similarity-Mueller et al(2015)[6]	Use of LSTM for text Similarity		Demonstration of how simple LSTM can perform complex Semantic similarity.

Table 2.1: Literature Survey

CHAPTER 3

SOFTWARE REQUIREMENTS SPECIFICATION

3.1 Assumptions and Dependencies

This section discusses the implicit as well as explicit assumptions that can be made without the loss of generality with respect to usage of the system in varied fields. The system assumes that the data set presented is coherent and does not include any contradiction in the text. Also, the presented data set will contain appreciable data so that the system can produce cogent output.

The system assumes that the complete data is in English. The system works comprehensively on English NLP, keeping in mind that the scope may be enhanced to Indic NLP in future.

3.2 Functional Requirements

Functional requirements are used to define a function of the system or a component of the system as a relationship between input and output. Functional requirements for the proposed system have been discussed in this section. All functional mainly revolve around these three points:

- Purpose - To suggest relevant questions for the entered query
- Input - A query or a question which is to be entered by the user
- Output - Set(s) of questions suggested for the entered query

The functional requirements for the system can be stated as follows:

3.2.1 Suggestion of questions

Suggest relevant Questions from the database as per the input query which pass some similarity threshold, also check the similarity threshold for the generated questions.

3.2.2 Collection of feedback

Record the feedback of the user and store the same efficiently for analysis in future. The input query can also be retained as the part of the data set.

3.2.3 Generating Reports

Generate reports about the performance of the system in order to determine the source of error in case of fatal errors, reports for doing further detailed analysis on the suggested questions and reports that help to graphically identify the performance of the system.

The required reports are:

- JSON files consisting details of recommended questions.
- Log files to asses functioning of the system and locate errors.
- Graphs to visualize the performance of the system.

The above requirements sum up the main functional requirements of the system. Addition in this chapter is possible as the scope of the system increases.

3.3 External Interface Requirements

This section discusses the need of external interface that is implemented in the system. External interface requirements are usually defined under four main subsections namely user, hardware, software and communication.

3.3.1 User Interfaces

In today's world user interface serves not only as a utility for users to interact with the system but also as a selling point for many applications across different domains and fields of work. The user must find the interface comfortable and navigation in the system should be intuitive and as per the general standards. The UI must provide functionality to the user in accordance with the level of access he/she has.

The UI must have the following characteristics:

- Comfortable to use
- Lucrative Design
- Easy to modify
- Intuitive to navigate
- Attractive to end user

The system will mainly be used by front end and back end user, the UI will provide functionality for back end users to operate on database and view the performance of the system.

Currently the system does not require any hardware or software interfaces, the same can be included later if need there be.

3.3.2 Communication Interfaces

Communication interfaces include email, messaging or network protocols that will be required by the system either to fulfill the requirement or provide a functionality. The system under consideration may be required to send periodical mails or files to system admin containing the reports regarding the functioning of the system.

For this system communications interface mainly contains reporting of the system performance reports to the system admin. The reports can be generated in JSON format which happens to easy to use, transport and work on.

3.4 Nonfunctional Requirements

Non-Functional Requirement (NFR) defines the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to success of the software system.

3.4.1 Performance Requirements

The system should be able to give a quick and efficient response for the entered query. The delay of response, if any, should be only due to lack of computation power and not due to inefficiency of the algorithm

3.4.2 Safety Requirements

- System should not crash due to overabundance of queries
- Abrupt failure of system should not happen

3.4.3 Security Requirements

- System should maintain a threshold of queries to be handled simultaneously so as to avoid getting into aborted state
- Addition of question to the database must be available for only those queries entered by the user i.e. direct addition to the database without displaying similar questions should not happen

3.4.4 Software Quality Attributes

The software quality attributes for the system under consideration are :

- Maintainability - Need of updation of database whenever query is entered
- Robust - The software should be able to handle multiple queries at a time
- Recoverability - The database should have replicas in case of loss of data
- Availability - User should be able to query the database anytime

3.5 System Requirements

This section deals with the requirements for the working of the system like Database Requirement, Software Requirements and Hardware Requirements.

- Database Requirements - The database is a BigQuery database, so the Google Cloud Platform (GCP) should be enabled for querying the database if required.
- Hardware Requirements
 - Intel i5 Processor
 - GPU for performing computations on large datasets
- Software Requirements
 - NLTK Library
 - Gensim Library
 - Python 3
 - Flask 1.0.2

3.6 Analysis Models: SDLC Model

SDLC stands for Software Development of Life Cycle and represents the methodology for efficient implementation that will be followed during completing the entire project. Different models like Waterfall, V-Model, Agile methodology were taken into consideration and Agile methodology has been chosen for the project as it best suits the nature of the project.

Agile model has six phases namely Planning, Analysis, Design, Implementation, Testing and Integration and Maintenance.

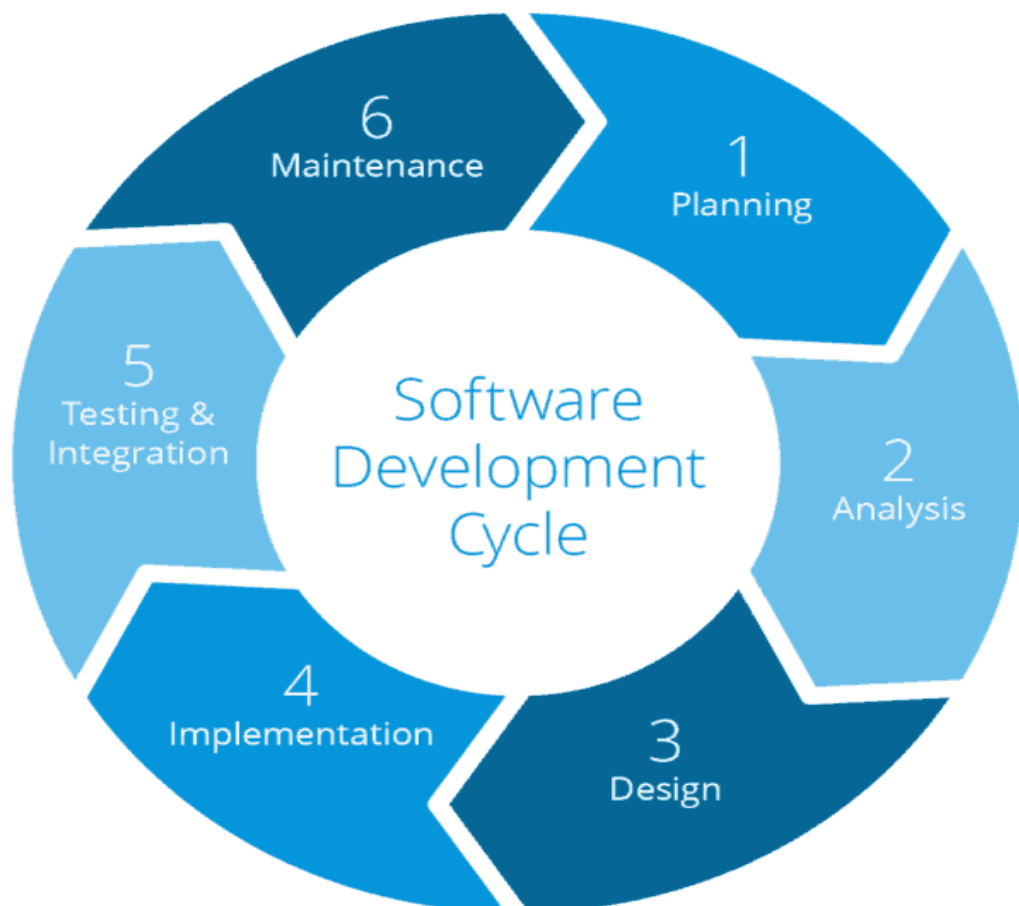


Figure 3.1: Agile Model

I. Planning - The outline of the project was prepared along with the deciding the timeline of the project

II. Analysis - The feasibility of the project and the expected outcome of the project was analyzed

III. Design - The process and workflow of the project were created in this phase

IV. Implementation - Actual programming of the project on the basis of the design laid was carried out. Overall, entire implementation of the system is completed.

V. Testing and Integration - The complete unit and integration testing of all the modules of the project was performed in this phase

VI. Maintenance - The developed project is maintained by performing regular checks on the working of the project and making sure that it produces the expected output

Agile model helps to assess the progress of the project at hand at periodic intervals thereby helping to judge the efficiency of the work even at early stages. Agile methodology also helps in avoiding the trickling of errors down the stream, as compared to Waterfall model and other similar models Agile methodology proves to be beneficial with respect to incorporating changes downstream.

Hence Agile methodology was used during the complete SDLC for the system under consideration.

CHAPTER 4

SYSTEM DESIGN

System design helps to graphically represent the system as a whole. This chapter discusses the system architecture consisting both the system hierarchy and detailed architecture for syntactic similarity, semantic similarity and combined approach for syntactic and semantic similarity.

4.1 System Architecture

The system architecture represents overall structure of the system. The system takes two inputs namely document database and query from the user and suggests similar questions from the existing database using syntactic and semantic similarity.

The system compares the input query with the questions present in the database and suggests relevant questions from the database. The system mainly has three different approaches to suggest relevant questions, the internal architecture remaining the same. The three approaches are as follows:

- Database Tagging - Pre-decided w.r.t the database whether to use syntactic similarity or semantic similarity.
- Static Lambda - Giving equal weightage to syntactic as well as semantic similarity.
- Self-Learning Lambda - Using both the similarities but with one having higher importance than the other.

The system architecture including the overview of the system along with the detailed architecture of different approaches is as follows:

- Overall Hierarchy
- Syntactic Similarity
- Semantic Similarity
- Database Tagging
- Combined Approach

4.1.1 Overall Hierarchy

The diagram represents the bird's eye view of the system:

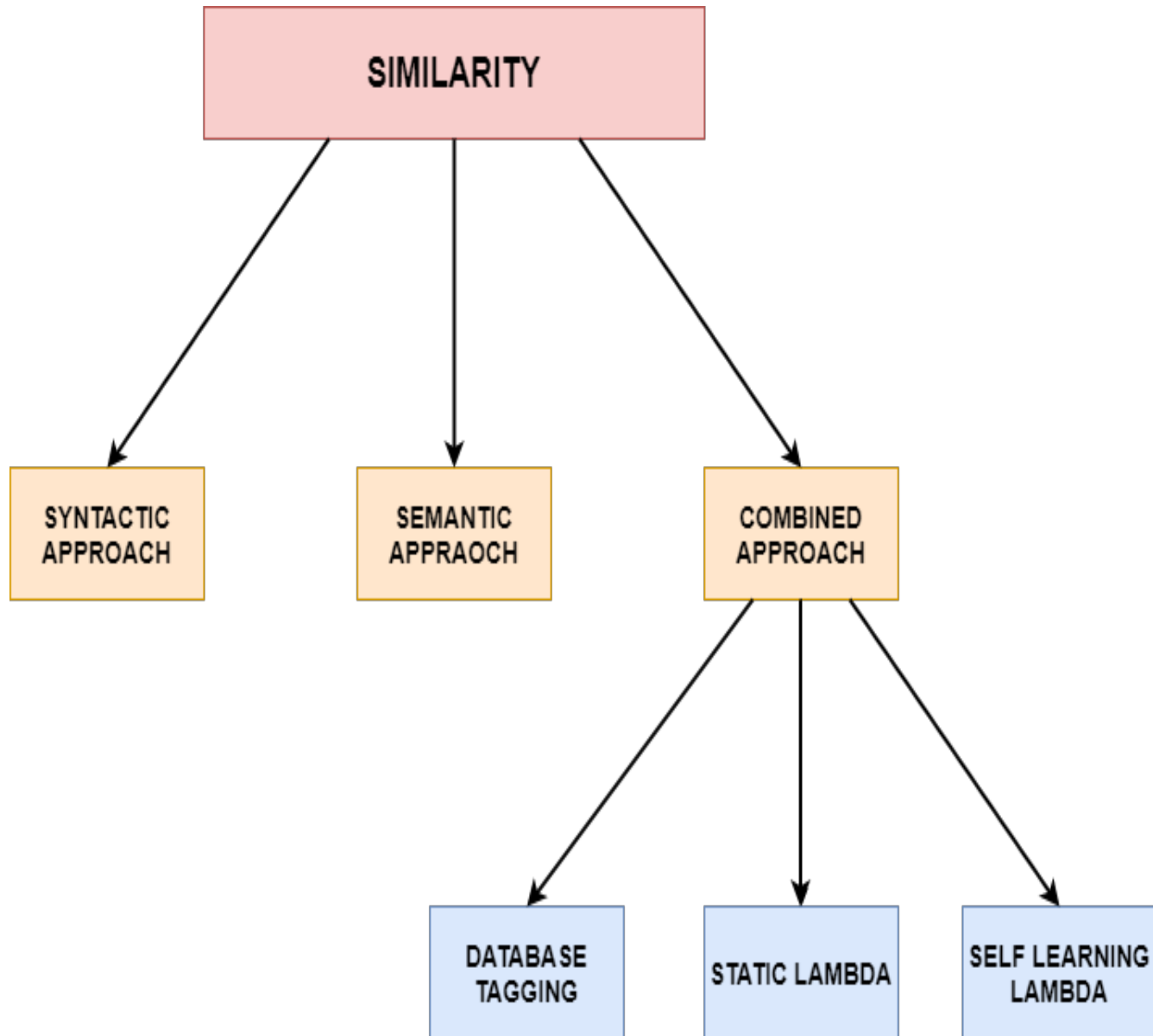


Figure 4.1: Bird's Eye View - Overview

4.1.2 Syntactic similarity architecture

The below presented architecture is used to find syntactic similarity, the system pre processes the data in the database and the input query by tokenising, removing stop words and punctuation and stemming the words. It then uses the cosine similarity to find the similarity index between two vectors.

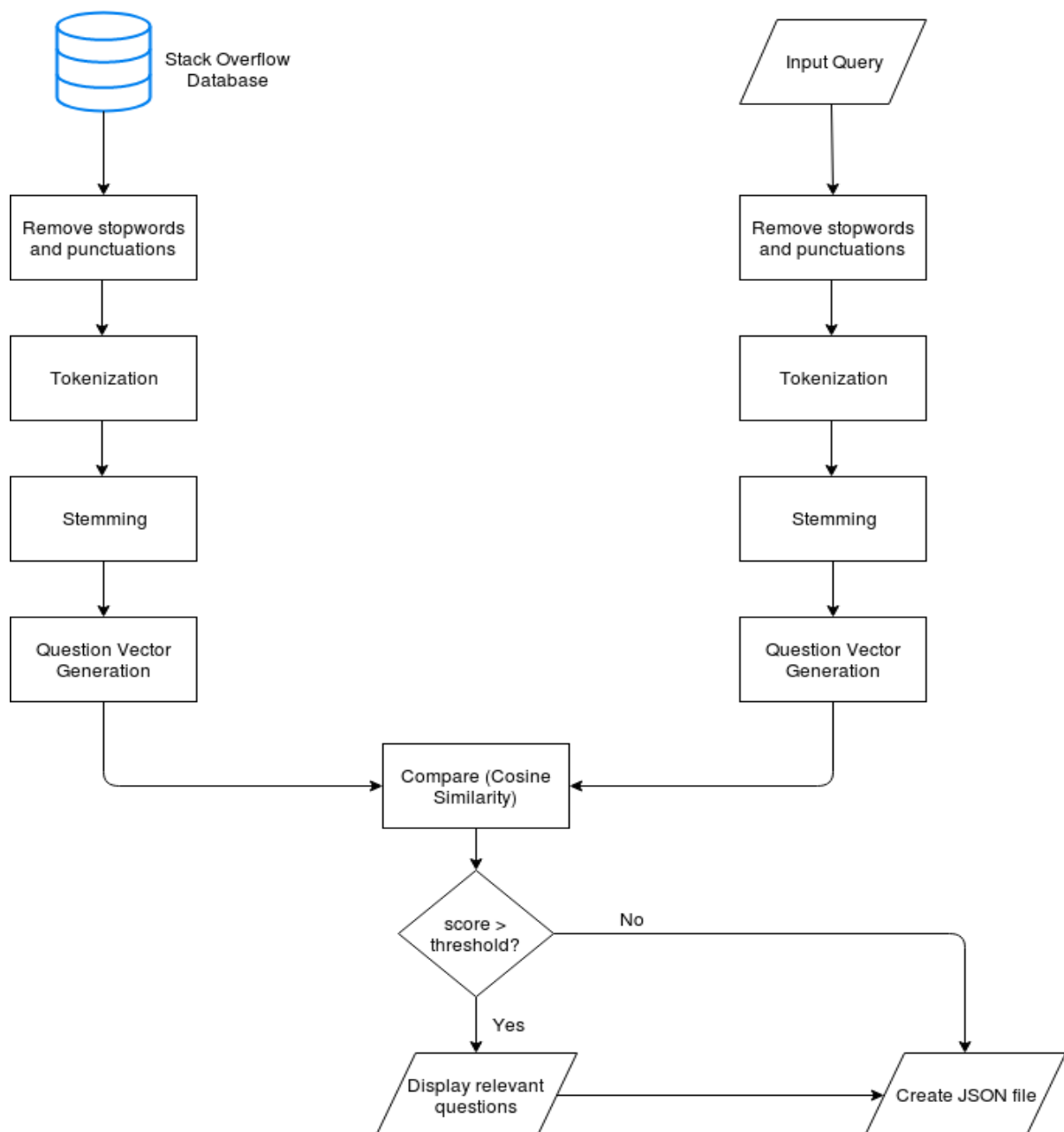


Figure 4.2: Syntactic Similarity - Cosine Similarity

4.1.3 Semantic Similarity Architecture

The doc2vec architecture is used for semantic similarity. It is similar to the popularly known word2vec. The only addition to the word2vec model is that a document or a paragraph ID is associated with the word vectors in the input layer. It then works similar to the word2vec model.

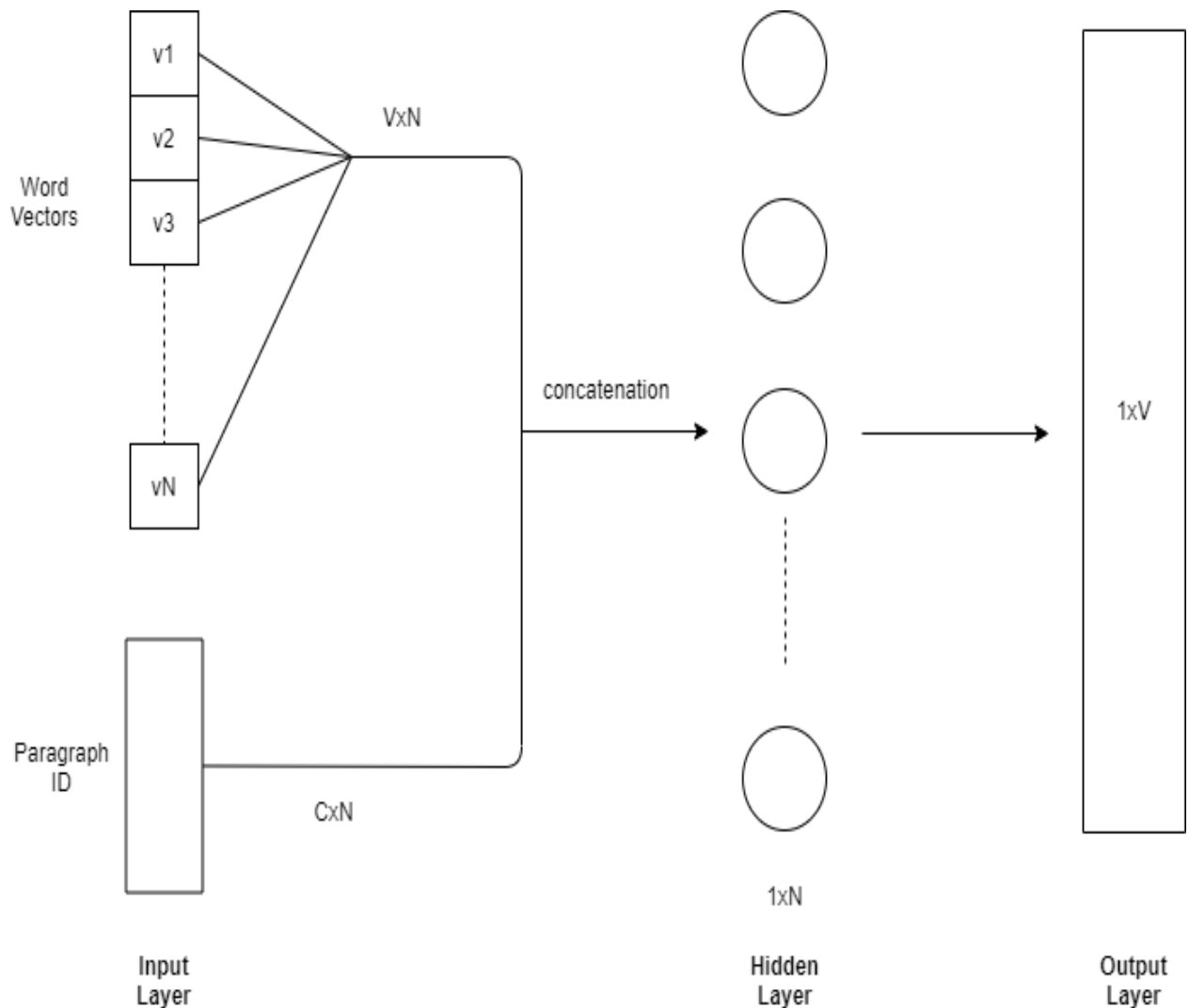


Figure 4.3: Doc2Vec Architecture

4.1.4 Database Tagging Architecture

This architecture is proposed for a use case wherein the user is sure of the similarity approach that would yield the best result. The system accepts as input a file that contains name of the database file and a tag indicating the approach to used.

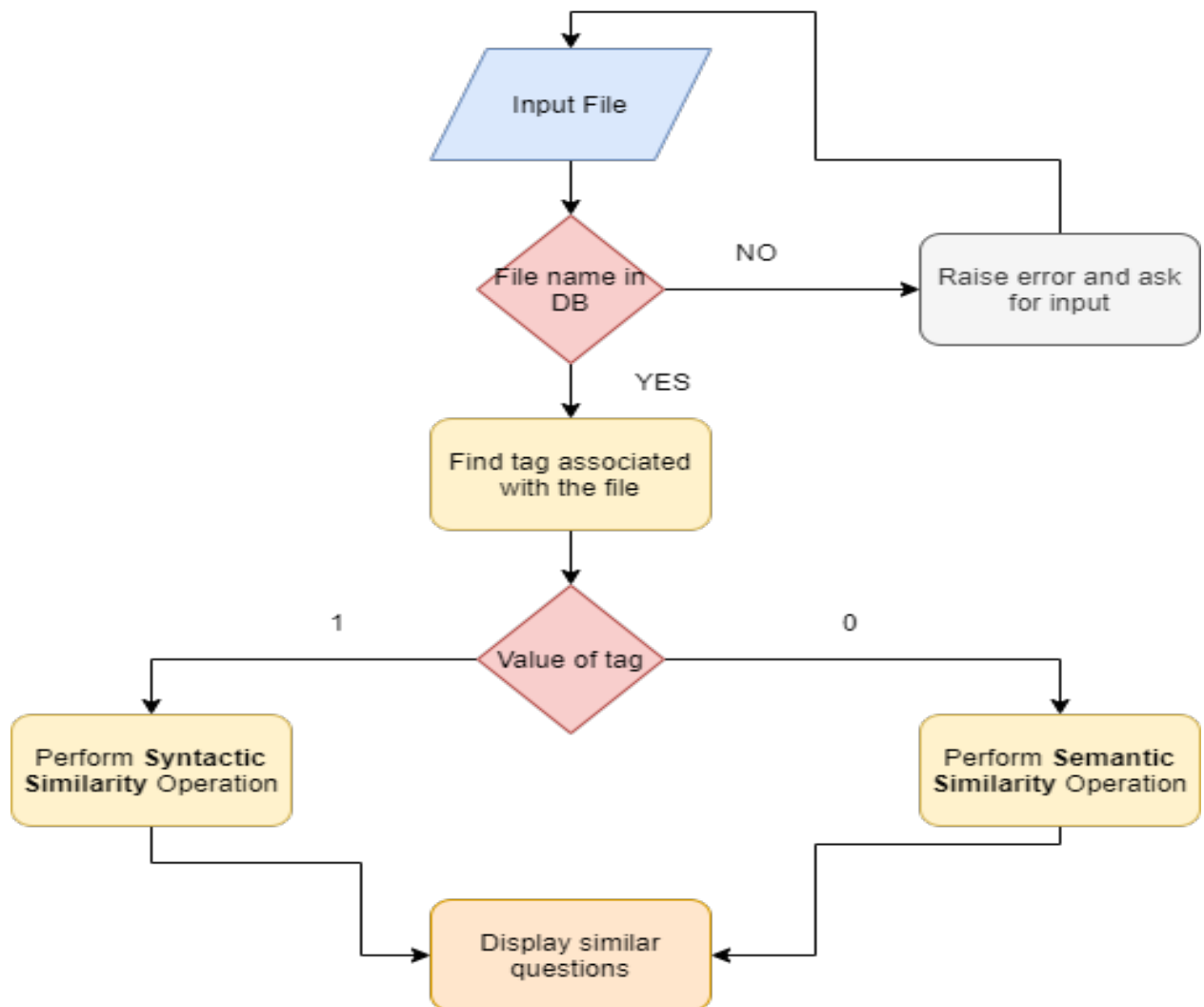


Figure 4.4: Database Tagging Architecture

4.1.5 Self Learning Lambda Architecture

The self learning architecture as presented below takes as input, multiple sets created from one database, where one set is formed by a query and a file that has similarity ranks according to the query. For different values of lambda the SSRD value is calculated by comparing the input ranks and the ranks generated by the system. The best value of lambda is obtained for the least value of SSRD. To obtain the value of lambda for the complete database the average of all the optimum lambda values for respective data sets is calculated.

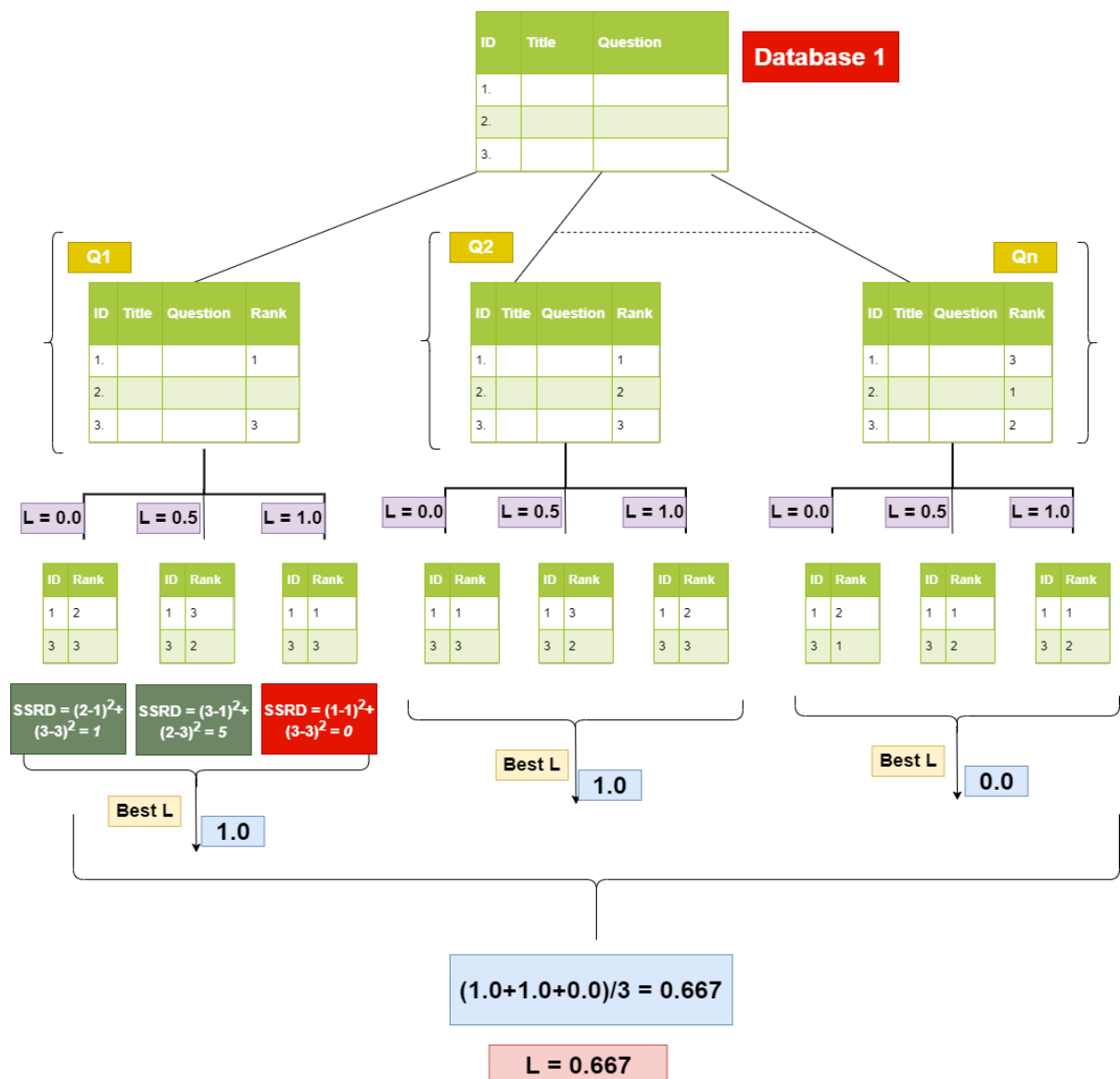


Figure 4.5: Self Learning Lambda Architecture

4.2 Data Flow Diagrams

Data flow diagrams help us to understand the how data is received, managed, stored and transported by the system. DFDs are usually discussed on three different levels, they also happen to be crucial in analysing any pitfalls if there exists any regarding data security in the system.

4.2.1 DFD Level 0

DFD Level 0 diagram represents the abstraction view which shows system as a single process with its relationship with external entities.

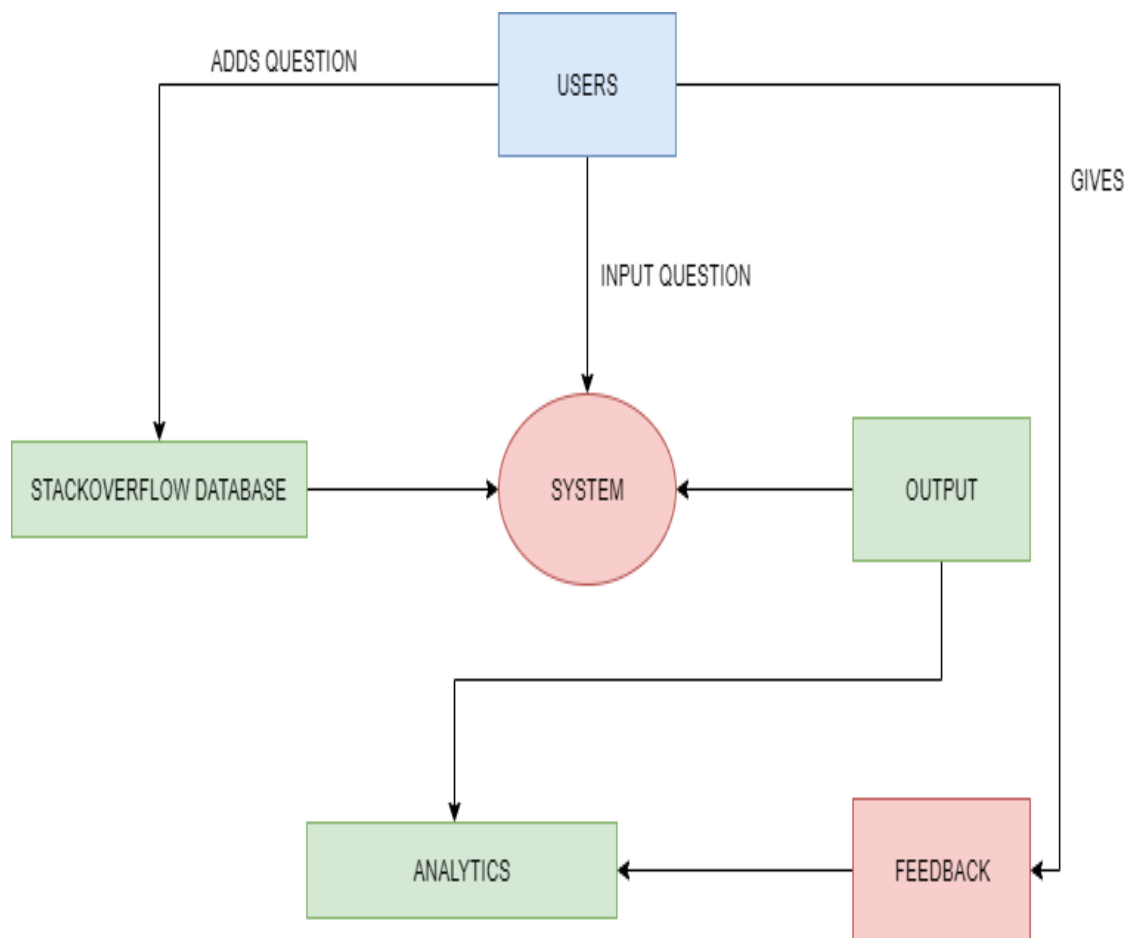


Figure 4.6: DFD Level 0

4.2.2 DFD Level 1

In this level we highlight the main functions of the system and breakdown the high level processes of 0-level DFD into subprocesses.

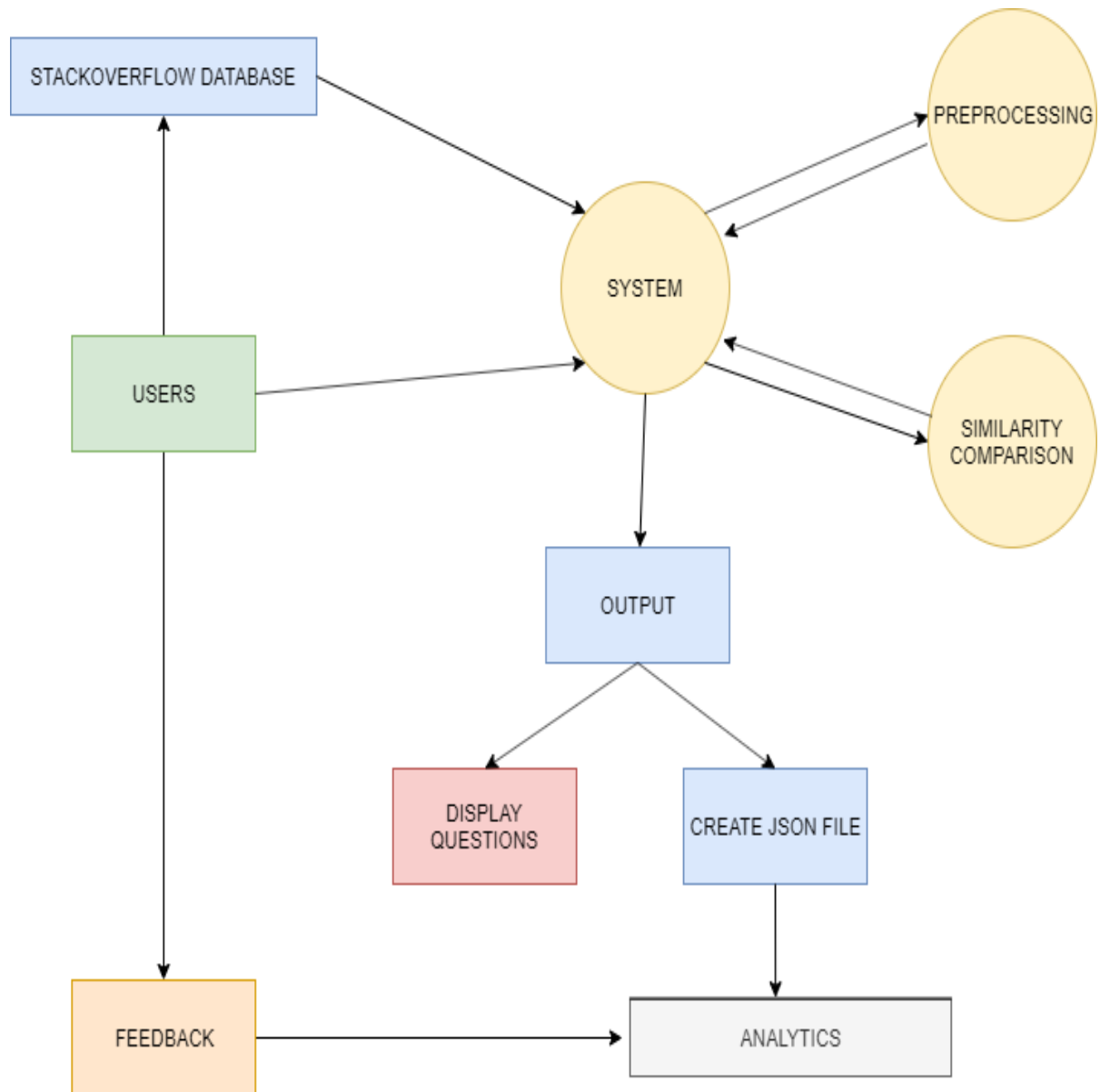


Figure 4.7: DFD Level 1

4.3 UML Diagrams

UML stands for Unified Modelling Language, it is widely used to understand the relationship between the external users and the system components, we can also understand the way users interact with the system using the UML diagram.

4.3.1 Use Case Diagram

Use case diagram consists of actors and use cases. The actors here are Front End User, System Level User and Administrator.

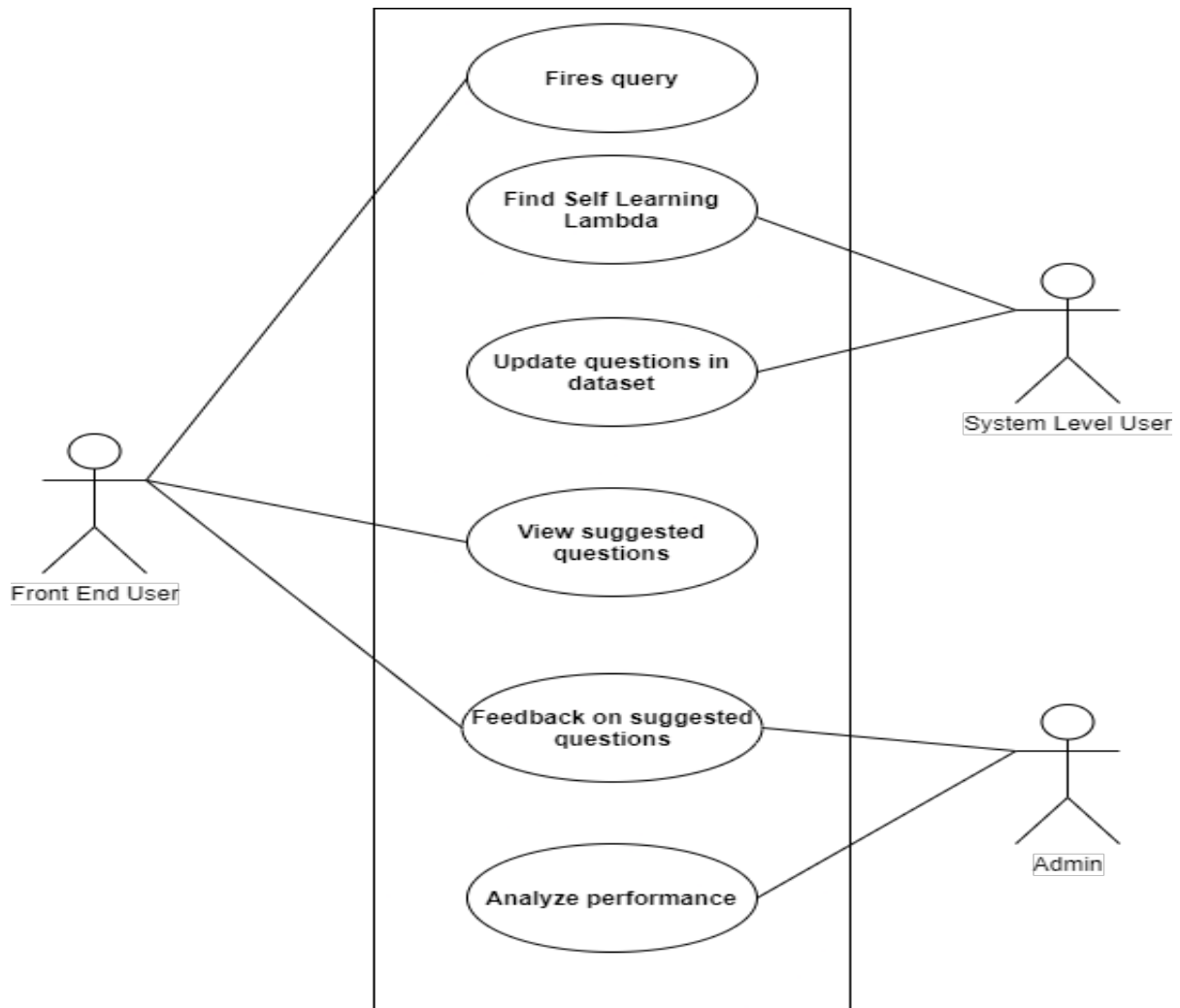


Figure 4.8: Use Case Diagram

CHAPTER 5

PROJECT PLAN

5.1 Project Estimate

Project estimate is an estimation of time and money invested to complete a project. The time required for the entire completion of the project was around 9 months. Any extra monetary expenses were not required as we had free GPU support available from our institute.

5.1.1 Reconciled Estimates

The initial time estimate for designing and implementing syntactic and semantic similarity was around 2 months. But we as a team then realized that the two types of similarities can be combined, and hence later changed the time estimate to 3 months and completed the combined approach in 3 months successfully.

5.1.2 Project Resources

The project resources include:

- NLTK Library
- Gensim Library
- GPU
- Flask Framework
- Large Dataset

5.2 Risk Management

Risk management deals with identifying the potential risks of the project in advance, analyzing them and taking precautionary measures to curb the risk.

5.2.1 Risk Identification

The project initially was proposed just to find similar questions. The process of finding similar questions was based on either syntactic similarity or semantic similarity. Which similarity technique worked better, was governed by the input data that was being used. The risk identified here was that we did not have a comprehensive method to choose the similarity technique. From the user's point of view, the built system should work coherently for different types of input data, with the system itself deciding to choose the type of similarity.

5.2.2 Risk Analysis

During the risk analysis phase, we brainstormed about the different scenarios where the identified risk could pose a threat. These included:

- Dataset of diverse fields
- Weak score of syntactic similarity on natural english language data viz. E-Commerce websites
- Weak score of semantic similarity on terminology specific data viz. Stack-Overflow dataset

5.2.3 Overview of risk mitigation, monitoring and management

- Mitigation - We mitigated the identified risk by devising a combined approach for syntactic and semantic similarity.
- Monitoring - We, as a team, constantly monitored the progress of the project by keeping in check that we are paying attention to the identified risks as well as any potential risk that may occur in future.
- Management - We successfully identified potential risks such as the integration of Flask for the user interface with the implemented machine learning models took precautions like performing modular development, and integrating the user interface right at the very end, in order to curb the risk of non-performance.

5.3 Project Schedule

The following section discusses the schedule that was followed for the implementation of the project, the division of the project into tasks and their timeline of execution is also presented below.

5.3.1 Project Task Set

Task set consists of software engineering tasks, milestones and deliverables. In our project, following is the task set:

- Software Engineering Tasks - These tasks present were viz. requirement gathering,
- Milestones - Milestones were viz. syntactically similar questions, semantically similar questions, overall similar questions, displaying similar questions through a UI and firing the similar questions on a search engine.
- Deliverables - The system for equal weight age to syntactic and semantic similarity, unequal weight age based on the learning for syntactic and semantic similarity and pre-tagged dataset for performing either syntactic or semantic similarity.

5.3.2 Task Network

The task network comprises of interlinked tasks. In simple words, it consists tasks that are dependent on each other. In this project, the task network had heavy interlinking between the task of syntactic and semantic similarity. There exists two critical paths for the proposed task network. The critical path of the task network goes like this: Start - Prepare Dataset - Syntactic Similarity/Semantic Similarity - User Interface for Display - Finish.

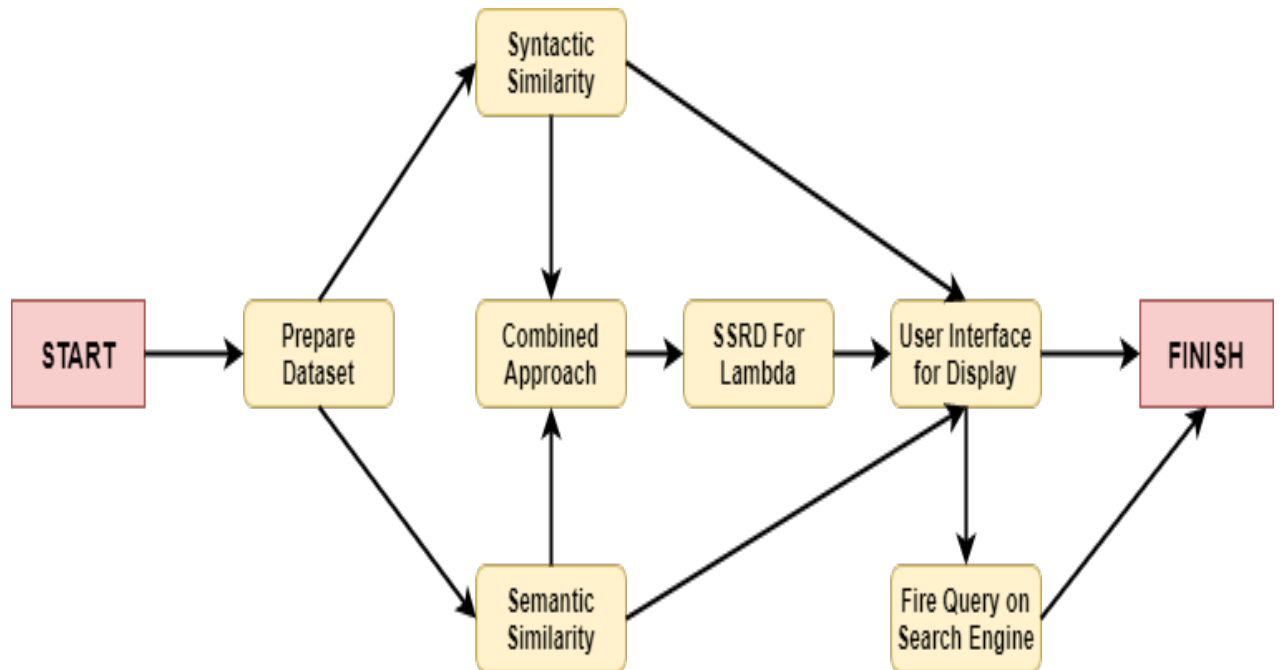


Figure 5.1: Task Network

5.3.3 Timeline Chart

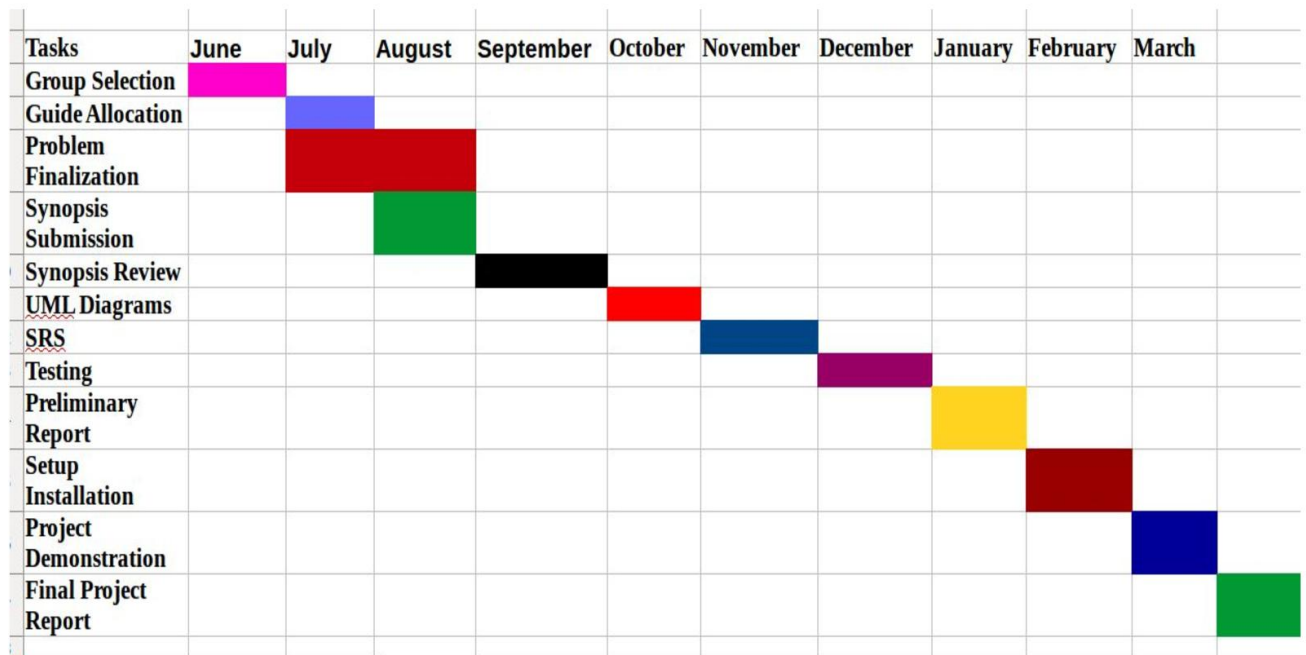


Figure 5.2: Timeline Chart

5.4 Team Organization

The project was completed by a team of three students each assuming different roles during the complete process, this section provides the team details and its working.

5.4.1 Team Structure

The team comprised of 3 members, with one taking the leading role in the overall system designing of the project. The other two were active members as well as contributors and provided timely suggestions and feedback to the leader.

5.4.2 Management reporting and communication

The entire team cooperated with each other throughout the building of the project.

- The cooperation was evident right from forming project requirements till maintaining the built project.
- We as a team, reported to our mentor in a timely manner and received invaluable inputs and guidance. Timely guidance and suggestions from our mentor was very crucial in the successful completion of our project.
- The communication between the team was very transparent. We always tried our best to help each other and would put forward our ideas and suggestions without hesitation. All in all, the communication amongst the team was nothing less than flawless.

CHAPTER 6

PROJECT IMPLEMENTATION

6.1 Overview of Project Implementation

The project implementation was carried out for the five systems and two of them were built from end to end to be presented as working products. The brief for each of the system are as below:

- Syntactic Similarity System

This system is built to as a proof of concept and not as a product, it receives a csv file which it stores as it's database, an input query and a stop words file. The main modules can be classified as

- Read data
- PreProcess Data
- Find syntactic similarity
- Display data

- Semantic Similarity System

The semantic system is a proof of the concept that semantic systems can accept questions that mean them even though they may not use the same wording, its functioning is quite similar to syntactic system, but in place of finding syntactic similarity it uses semantic similarity. The main modules can be classified as

- Read data
- PreProcess data
- Find semantic similarity
- Display data

- Database Tagging System

This system has been built from end to end with the intention of presenting it as a complete product. The system find the tag associated with the file and uses one similarity approach either syntactic or semantic depending upon the tag. Modules are as follows

- Read tag file data
- Find Syntactic Similarity
- Find Semantic similarity
- Display data on UI

- Static Lambda System

This system has also been developed end to end and it uses a combined approach towards finding similar questions from the database. This system produces better results as compared to the systems which use a single approach for finding similarity. The main modules can be classified as

- Read data
- Find combined similarity
- Display data on UI

- Self Learning Lambda System

This system is supposed to be used in compliance with the static lambda system, it finds the optimum value of lambda, which is an indicator of the importance given to each similarity technique to find similar questions. This system is aimed at analysing the database and is not to be used by the end user. The modules are

- Read driver file data
- Finding optimum value of lambda
- Store the obtained results

6.2 Tools and Technology Used

The tools and technology used in the project are :

1. X86-64 Architecture
2. Intel i5-7200 CPU
3. 512 GB SSD, 8GB RAM
4. Python 3.6.9
5. NLTK Library 3.4.5
6. Matplotlib Library 3.1.1

6.3 Algorithms Details

Algorithms used for syntactic similarity, semantic similarity operation, DB tagging system and the self learning system have been presented in this section.

6.3.1 Algorithm for Syntactic Analysis

The code written to implement the below mentioned algorithm has been implemented in python, Pseudo Code used for the basic implementation is as follows :

procedure SyntacticSimilarityOperation(input_query,similarity_threshold,Question_Database)

```
1 list_of_questions := read(Question_Database)

2 list_of_stop_words := read(Stopwords_File)

3 processed_list := PreProcess(list_of_stop_words)

4 input_query := read(Input_from_User)

5 for question in processed_list

    5.1 similarity_value := calculate_cosine_similarity(input_query,question)
    5.2 if similarity_value  $\geq$  similarity_threshold then
        5.2.1 print(question)
    5.3 endif

6 endfor

7 endprocedure SyntacticSimilarityOperation
```

procedure PreProcess(list_of_questions,list_of_stop_words)

```
1 for question in list_of_questions

    1.1 question := tokenize(question)
    1.2 question := nltk.PorterStemmer.stem(question)

2 endfor

3 return list_of_questions

4 endprocedure PreProcess
```

Formula used for calculating cosine similarity is as follows:

$$\cos(\mathbf{t}, \mathbf{e}) = \frac{\mathbf{t} \cdot \mathbf{e}}{\|\mathbf{t}\| \|\mathbf{e}\|} = \frac{\sum_{i=1}^n \mathbf{t}_i \mathbf{e}_i}{\sqrt{\sum_{i=1}^n (\mathbf{t}_i)^2} \sqrt{\sum_{i=1}^n (\mathbf{e}_i)^2}} \quad (6.1)$$

6.3.2 Algorithm for Semantic Analysis

The algorithm used for semantic analysis uses the doc2vec model. The pseudo code for the implementation of the same is as follows:

procedure SemanticSimilarity(word2vec_model, input_question, stopwords_file)

```
1 stopwords_list := generate_list(stopwords_file)
2 ds := Doc2Vec(word2vec_model, stopwords_list)
3 list_of_ques := ds.CalcSimilarity(input_question, list_of_ques)
4 return list_of_ques
5 endprocedure SemanticSimilarity
```

procedure CalcSimilarity(input_question, list_of_ques)

```
1 src_vec := vectorize(input_question)
2 for ques in list_of_ques
    2.1 ques.vector := vectorize(ques.question)
    2.2 ques.similarity_score := float(cosine_sim(src_vec, ques.vector))
3 list_of_ques.sort(reverse := True)
4 return list_of_ques
5 endprocedure
```


6.3.3 Algorithm for Database tagging

The algorithm used for deciding the type of operation to be performed on a given database file depending on the tag mentioned in tag file is discussed below, this algorithm requires a tag file as input, each line in tag file contains the name of a file and its corresponding tag, either 0 or 1.

procedureDetermine_DB(input_file_name,tag_file)

```
1 if file_name in tag_file then
    1.1 tag = file_name.tag
    1.2 if tag = 1
        1.2.1 syntacticsimilarity(input_file_name)
    1.3 else
        1.3.1 semanticsimilarity(input_file_name)
    1.4endif
2 else
    2.1 raise error
3 endif
```

6.3.4 Algorithm for Self Learning Lambda

The following algorithm is used to analyse the Database and find the optimum value of lambda for the database under consideration, it accepts a driver file as input, which contains q query question and a file in which the questions are ranked according to the query.

procedure Determine_optimum_Lambda(driver_file)

```
1 for query,file_name in driver_file

    1.1 find SyntacticSimilarity(query,file_name)

    1.2 find SemanticSimilarity(query,file_name)

    1.3 for lambda in 0.0 -> 1.0

        1.3.1 for question in file
            1.3.1.1 question.similarity := combined_similarity
        1.3.2 endfor

    1.4 sort(list_of_questions) in decreasing order

    1.5 initialize ssrd := 0

    1.6 for question in file

        1.6.1 if question.rank is present then
            1.6.1.1 ssrd := ssrd + (predicted_rank - question.rank)2
        1.6.2 endif

    1.7 store(lambda,ssrd)

    1.8 endfor

    1.9 optimum_lambda_for_file = lambda where ssrd is minimum

2 sum_lambda := sum of all obtained optimum_lambda_for_file

3 best_lambda := sum_lambda/number_of_sets in driver_file
```

Formula used for calculating combined similarity is as follows:

$$\text{combined_similarity} = \lambda * \text{syntactic_similarity_score} + (1 - \lambda) * \text{semantic_similarity_score} \quad (6.2)$$

CHAPTER 7

SOFTWARE TESTING

Software Testing is the activity which helps us to compare the actual results and the expected results. It also ensures that the system is free of defect. It also helps to identify the errors or missing requirements which were a part of the original requirements.

7.1 Types Of Testing

Testing is of the following three types :

1. Functional Testing
2. Non-Functional Testing

7.1.1 Functional Testing

Functional Testing helps us to test against the Functional Specifications. Functional Testing involves the following steps :

- Identification of the function of the software.
- Create input database on the function's specification.
- Determine the output based on the specification of the function.
- Execute the test case.
- Compare the actual and expected outputs.

The system was unit tested as well as integration tested.

7.1.2 Non-Functional Testing

Non-Functional Testing helps us to test the Non-Functional aspects like performance, usability, reliability etc.

Characteristics of Non-Functional Testing :

- It should be measurable i.e., there is no place for subjective characterization.
- It ensures the quality attributes of the system.

The system is tested on Reliability, Usability, Portability, Flexibility and Re-usability.

7.2 Test Cases And Test Results

Intensive testing was carried on the system to determine loop holes and system's behaviour in extreme conditions. Each subsection displays a subset of the testing records manually maintained.

7.2.1 Unit Testing

Unit Testing is a type of Functional Testing in which individual components of the system are tested. It's purpose is to validate each unit of the system. It usually has one or more input value and a single output value.

1	Unit	Test Input	Output	Result
2				
3	Preprocessing	list of questions (What is use of python ?), stopwords list	preprocessed list of questions(what use python)	PASS
4	Preprocessing	empty list	empty list	PASS
5	Calculate Cosine similarity	input question vector (use python), list of DB question vectors (use java)	similarity value between 0 and 1 (0.5)	PASS
6	Calculate Cosine similarity	empty list	throw Error	PASS
7	Semantic similarity	Input question (How old are you ?), list of questions (What is your age ?)	similarity value between 0 and 1 (0.9)	PASS
8	Semantic similarity	empty list	throw Error	PASS
9	Read file	file name	list of questions in appropriate class	PASS
10	Read file	invalid name("asvasd")	Log file : Unable to read "asvasad"	PASS
11	Read Tag File	file name	file name associated with tag	PASS
12	Read Tag File	invalid name("asvasd")	Log file : Unable to read "asvasad"	PASS
13	Read Driver file	file name	file name with associated query in appropriate class	PASS
14	Read Driver file	invalid name("asvasd")	Log file : Unable to read "asvasad"	PASS
15	generate Graph	Similarity Percentage and Frquency	Bar graph save in mentioned location	PASS
16	generate Json	Similarity Percentage, Question and query	Json file mentioned location	PASS
17	create link	Question (what is python ?)	search engine link (https://duckduckgo.com/?q=what+is+python)	PASS

Figure 7.1: Unit Testing

7.2.2 Integration Testing

Integration Testing is a type of Functional Testing in which we test the system as a whole. We integrate the individually tested units and test the interactions between them.

It's purpose is to expose defects in the interface and the interactions between the integrated components.

1	System	Test Input	Output	Result
2				
3	Syntactic similarity	Database file name, input question	list of similar questions on console, json file, log file, similarity graph	PASS
4	Syntactic similarity	Non existent file name ("ABC"), input question	log file : Unable to open file "ABC"	PASS
5	Semantic Similarity	Database file name, input question	list of similar questions on console, json file, log file, similarity graph	PASS
6	Semantic Similarity	Non existent file name ("ABC"), input question	log file : Unable to open file "ABC"	PASS
7	Database Tagging	Tag file name, input question	list of similar questions on UI, json file, log file, similarity graph	PASS
8	Database Tagging	Non existent file name ("ABC"), input question	Raise error ask for input again on console	PASS
9	Self Learning Lambda	Database file name	optimum value of lambda on console, observation file, analysis graphs	PASS
10	Self Learning Lambda	Non existent file name ("ABC")	log file : Unable to open file "ABC"	PASS
11	Static Lambda	Database file name	list of similar questions on UI, json file, log file, similarity graph	PASS
12	Static Lambda	Non existent file name ("ABC")	log file : Unable to open file "ABC"	PASS

Figure 7.2: Integration Testing

7.2.3 Stress Testing

For a system it must be ascertained that it functions in accordance with acceptable limits of performance, to assess the same stress testing was performed on the system. CPU was used instead of a GPU for the semantic system to assess the system's performance in limited resources.

1	System	Input	Output	Result
2				
3	Syntactic Similarity	26K questions in data file	Result generated with lag of 16.486 sec	PASS
4	Semantic Similarity	26K questions in data file	Result generated with lag of 83.638 sec	PASS
5	Database Tagging System (Syntactic DB)	26K questions in data file	Result generated with lag of 16.621 sec	PASS
6	Database Tagging System (Semantic DB)	26K questions in data file	Result generated with lag of 83.837 sec	PASS
7	Static Lambda	26K questions in data file	Result generated with lag of 90.337 sec	PASS
8	Self Learning Lambda	8 files in driver files	Result generated with lag of 102.337 sec	PASS

Figure 7.3: Stress Testing

CHAPTER 8

RESULTS

8.1 Outcomes

The project has been divided mainly into three parts on the basis of the approach used namely syntactic, semantic and combination of both. The main outcomes of the project are listed below:

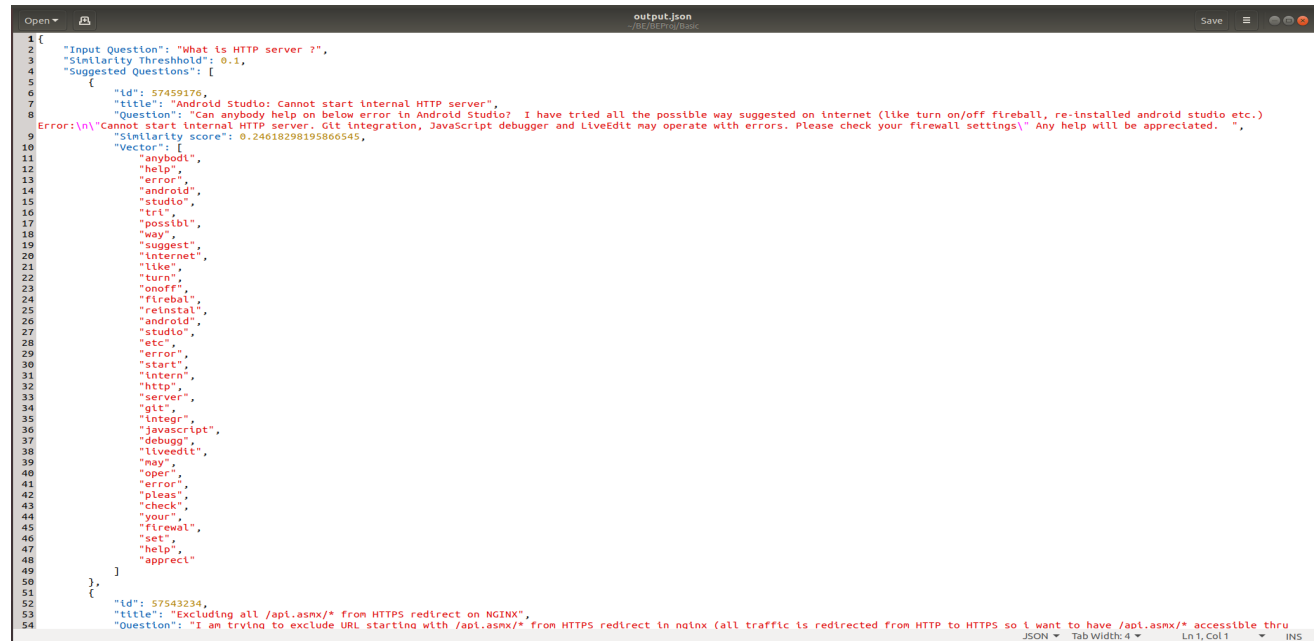
- Implementation of stand alone Syntactic System, analysing its benefits and limitations.
- Implementation of stand alone Semantic System, analysing its benefits and limitations.
- Developing Database Tagging Product, with User Interface and log files.
- Analysing a combined approach and implementing self learning lambda system for the same.
- Developing Static Lambda System with User Interface and log files.

The Static Lambda system can work as a purely syntactic or purely semantic system depending on the value of lambda 1 or 0 respectively, but this system in such a case does not save any computational time because for any value of lambda both syntactic and semantic similarities are calculated. On the other hand the Database tagging system performs only one type of operation syntactic or semantic depending on the tag, thereby saving computational time. Hence if the self learned value of lambda is near 0 or near 1 that is near the extremes then it is better to use the DB tagging system which will provide with faster response as compared to the static lambda system.

All the results presented below are obtained on a CPU system with the following hardware specifications:

- X86_64 architecture
- 8 GB RAM, 512 GB SSD
- Ubuntu OS 18.04

8.2 Screen Shots

A screenshot of a code editor window titled 'output.json'. The editor shows a JSON object with the following structure:

```
{  "Input Question": "What is HTTP server ?",  "Similarity Threshold": 0.1,  "Suggested Questions": [    {      "id": 57459176,      "title": "Android Studio: Cannot start internal HTTP server",      "Question": "Can anybody help on below error in Android Studio? I have tried all the possible way suggested on internet (like turn on/off fireball, re-installed android studio etc.) Error:\n\n\"Cannot start internal HTTP server. Git integration, JavaScript debugger and LiveEdit may operate with errors. Please check your firewall settings\" Any help will be appreciated. ",      "Similarity score": 0.24618298195866545,      "Vector": [        "anybodt",        "help",        "error",        "android",        "studio",        "trt",        "possibl",        "way",        "suggest",        "internet",        "like",        "turn",        "onoff",        "firebal",        "reinstal",        "android",        "studio",        "etc",        "error",        "start",        "intern",        "http",        "server",        "git",        "integr",        "javascript",        "debugg",        "liveedit",        "may",        "oper",        "error",        "pleas",        "check",        "your",        "firewal",        "set",        "help",        "apprecat"      ]    },    {      "id": 57543234,      "title": "Excluding all /api.asmx/* from HTTPS redirect on NGINX",      "Question": "I am trvng to exclude URL starting with /api.asmx/* from HTTPS redirect in nginx (all traffic is redirected from HTTP to HTTPS so i want to have /api.asmx/* accessible thru"    }  ]}
```

 The code is color-coded, with strings in red and numbers in blue. The editor interface includes a top bar with 'Open', 'Save', and window control icons. The bottom status bar shows 'JSON', 'Tab Width: 4', 'Ln 1, Col 1', and 'INS'.

Figure 8.1: JSON Output

The above figure shows the generated JSON File. The JSON File shows the input question, similarity score, similar questions and other relevant details, the json format is easy to transfer and read. This file can be used for further analysis of the suggested questions.

Suggesting Relevant Questions For A Query Using Natural Language Processing Techniques

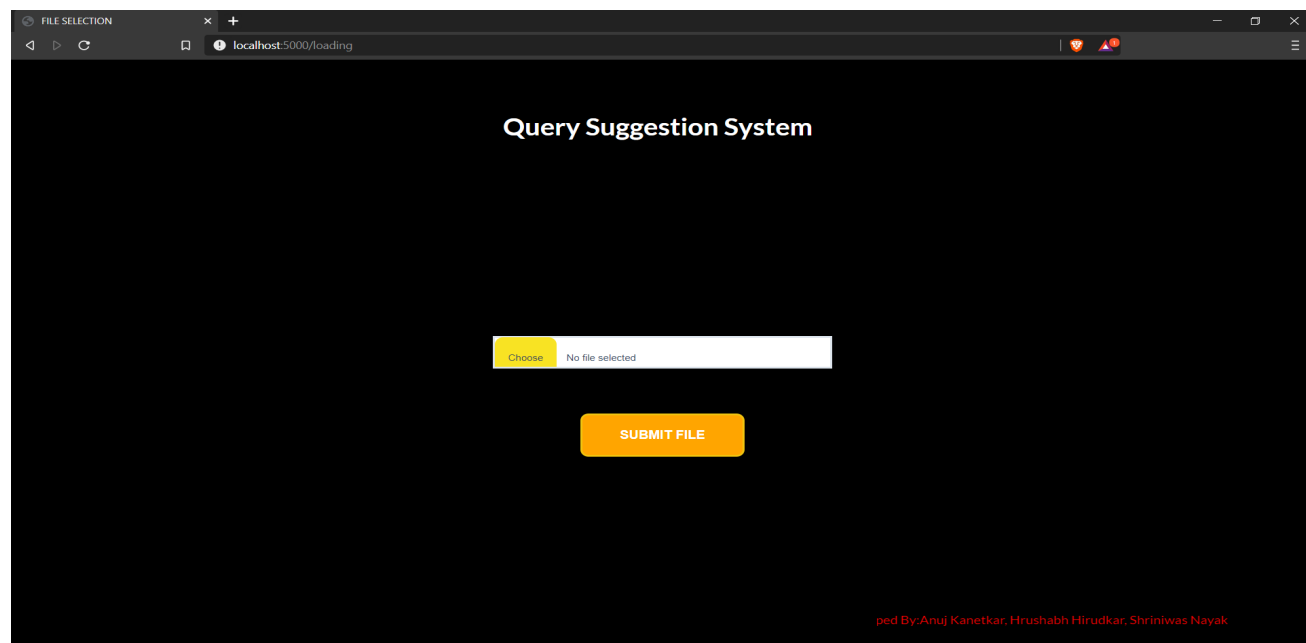


Figure 8.2: User Interface - File Selection

The User Interface shown above is of the File Selection page. This page allows the user to select the database on which he wants to perform the search. The User Interface allows easy reversal of actions and is intuitive for the user to navigate.

Suggesting Relevant Questions For A Query Using Natural Language Processing Techniques

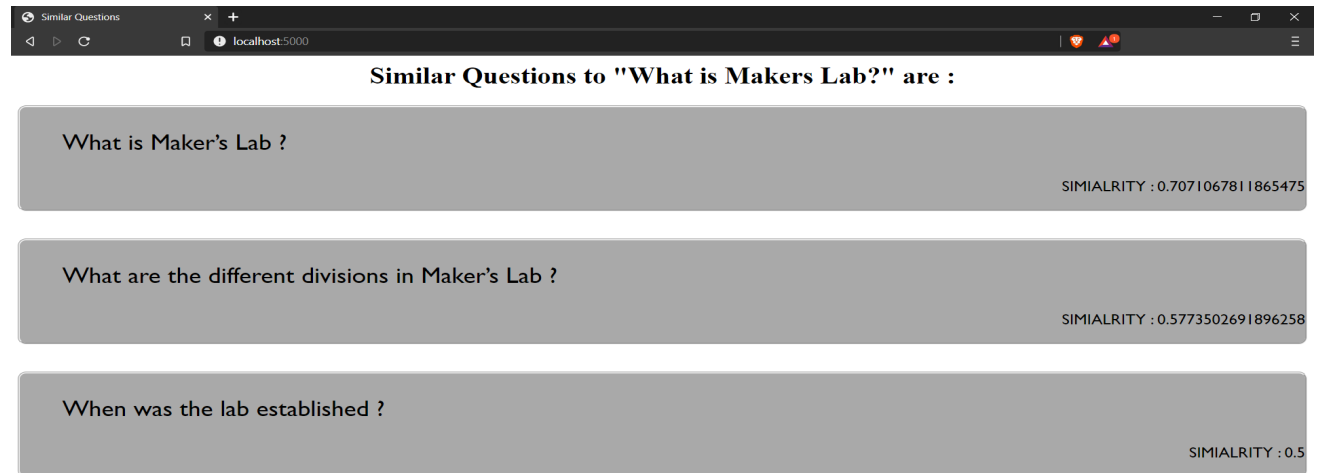


Figure 8.3: User Interface - Answers

This figure shows the User Interface of the Suggested Similar Questions. The user can click on any of the question which will lead him/her to a search engine where the question will be searched. Suitable static links can also be provided in the database file.

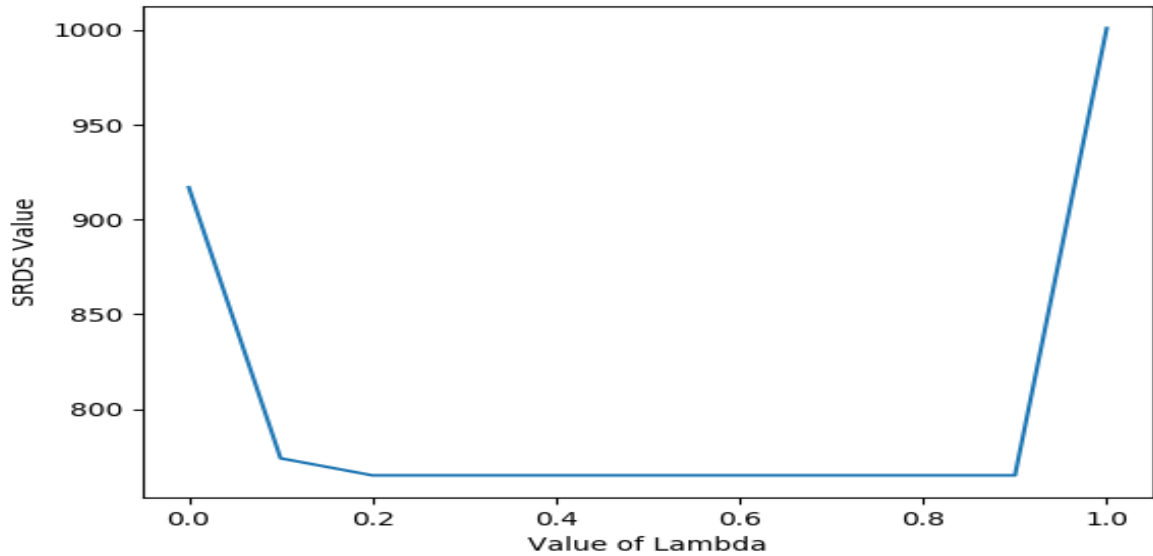


Figure 8.4: SSRD vs Lambda Graph

The plot represents the relation between the value of lambda which ranges from 0.0 to 1.0 and the error term, it can be learnt from this graph that the error term spikes at both the extremities indicating that a combined approach proves to be better for this case.

```

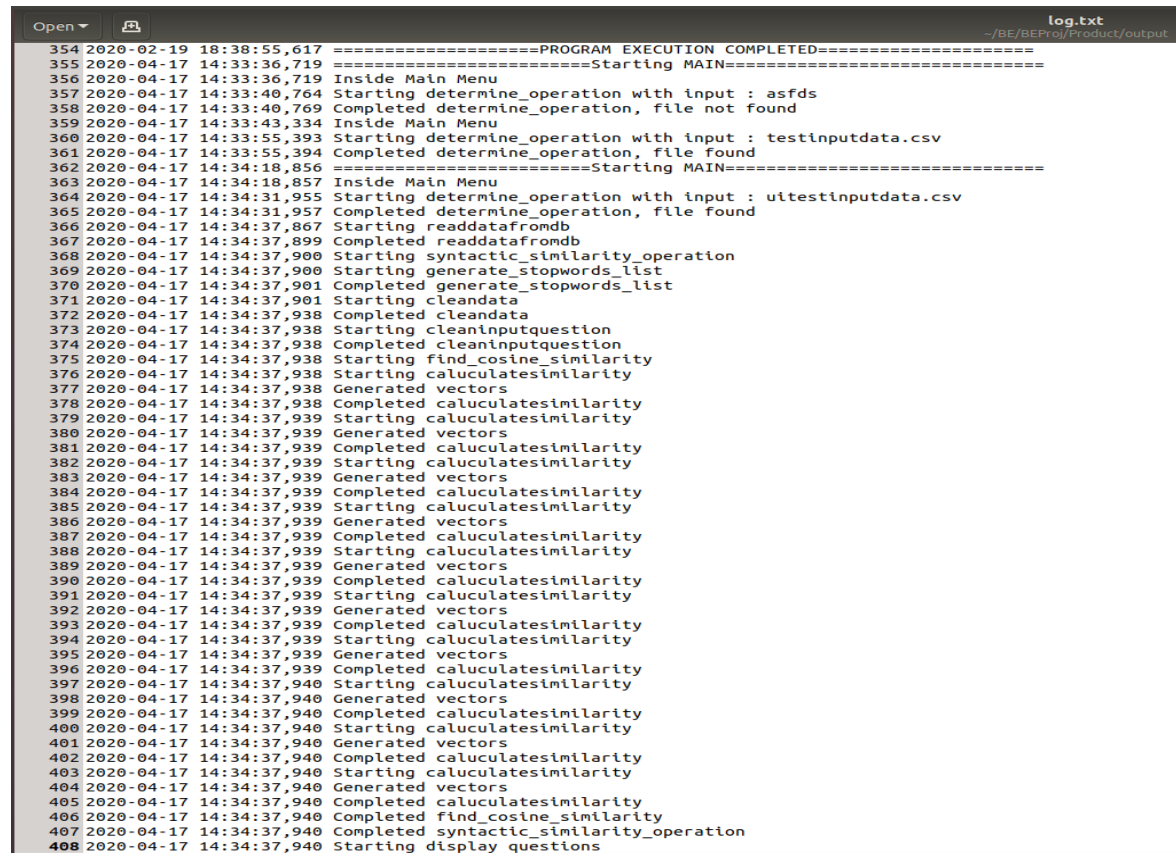
115
116
117 ::::::::::::::: BEST VALUE OF L : 1.000
118
119 =====
120
121 =====
122
123 File name : testoneinputdata2.csv
124 Query : Who started the Veturino project ?
125
126
127
128 L : 0.000    SRDS : 917.0
129 L : 0.100    SRDS : 774.0
130 L : 0.200    SRDS : 765.0
131 L : 0.300    SRDS : 765.0
132 L : 0.400    SRDS : 765.0
133 L : 0.500    SRDS : 765.0
134 L : 0.600    SRDS : 765.0
135 L : 0.700    SRDS : 765.0
136 L : 0.800    SRDS : 765.0
137 L : 0.900    SRDS : 765.0
138 L : 1.000    SRDS : 1001.0
139
140
141 ::::::::::::::: BEST VALUE OF L : 0.200
142
143 =====
144
145 =====
146
147 File name : testoneinputdata3.csv
148 Query : When will I get my first salary ?
149
150
151
152 L : 0.000    SRDS : 123.0
153 L : 0.100    SRDS : 123.0
154 L : 0.200    SRDS : 123.0
155 L : 0.300    SRDS : 123.0
156 L : 0.400    SRDS : 123.0
157 L : 0.500    SRDS : 123.0
158 L : 0.600    SRDS : 123.0
159 L : 0.700    SRDS : 123.0
160 L : 0.800    SRDS : 123.0
161 L : 0.900    SRDS : 123.0
162 L : 1.000    SRDS : 3.0
163
164
165 ::::::::::::::: BEST VALUE OF L : 1.000
166
167 =====
168
169 =====

```

Figure 8.5: Observation File

The above figure represents the observation file which is generated for the self learning system for finding the optimum the value of lambda. The Observation File shows the value of lambda and the SSRD that corresponds with it for multiple iterations and multiple query and file sets.

Suggesting Relevant Questions For A Query Using Natural Language Processing Techniques



The image shows a screenshot of a log file named 'log.txt' located at the path '~/BE/BEProj/Product/output'. The log contains a series of entries, each with a line number, a timestamp, and a description of the operation. The entries are as follows:

Line	Timestamp	Operation
354	2020-02-19 18:38:55,617	=====PROGRAM EXECUTION COMPLETED=====
355	2020-04-17 14:33:36,719	=====Starting MAIN=====
356	2020-04-17 14:33:36,719	Inside Main Menu
357	2020-04-17 14:33:40,764	Starting determine_operation with input : asfds
358	2020-04-17 14:33:40,769	Completed determine_operation, file not found
359	2020-04-17 14:33:43,334	Inside Main Menu
360	2020-04-17 14:33:55,393	Starting determine_operation with input : testinputdata.csv
361	2020-04-17 14:33:55,394	Completed determine_operation, file found
362	2020-04-17 14:34:18,856	=====Starting MAIN=====
363	2020-04-17 14:34:18,857	Inside Main Menu
364	2020-04-17 14:34:31,955	Starting determine_operation with input : uitestinputdata.csv
365	2020-04-17 14:34:31,957	Completed determine_operation, file found
366	2020-04-17 14:34:37,867	Starting readdatafromdb
367	2020-04-17 14:34:37,899	Completed readdatafromdb
368	2020-04-17 14:34:37,900	Starting syntactic_similarity_operation
369	2020-04-17 14:34:37,900	Starting generate_stopwords_list
370	2020-04-17 14:34:37,901	Completed generate_stopwords_list
371	2020-04-17 14:34:37,901	Starting cleandata
372	2020-04-17 14:34:37,938	Completed cleandata
373	2020-04-17 14:34:37,938	Starting cleaninputquestion
374	2020-04-17 14:34:37,938	Completed cleaninputquestion
375	2020-04-17 14:34:37,938	Starting find_cosine_similarity
376	2020-04-17 14:34:37,938	Starting caluculatesimilarity
377	2020-04-17 14:34:37,938	Generated vectors
378	2020-04-17 14:34:37,938	Completed caluculatesimilarity
379	2020-04-17 14:34:37,939	Starting caluculatesimilarity
380	2020-04-17 14:34:37,939	Generated vectors
381	2020-04-17 14:34:37,939	Completed caluculatesimilarity
382	2020-04-17 14:34:37,939	Starting caluculatesimilarity
383	2020-04-17 14:34:37,939	Generated vectors
384	2020-04-17 14:34:37,939	Completed caluculatesimilarity
385	2020-04-17 14:34:37,939	Starting caluculatesimilarity
386	2020-04-17 14:34:37,939	Generated vectors
387	2020-04-17 14:34:37,939	Completed caluculatesimilarity
388	2020-04-17 14:34:37,939	Starting caluculatesimilarity
389	2020-04-17 14:34:37,939	Generated vectors
390	2020-04-17 14:34:37,939	Completed caluculatesimilarity
391	2020-04-17 14:34:37,939	Starting caluculatesimilarity
392	2020-04-17 14:34:37,939	Generated vectors
393	2020-04-17 14:34:37,939	Completed caluculatesimilarity
394	2020-04-17 14:34:37,939	Starting caluculatesimilarity
395	2020-04-17 14:34:37,939	Generated vectors
396	2020-04-17 14:34:37,939	Completed caluculatesimilarity
397	2020-04-17 14:34:37,940	Starting caluculatesimilarity
398	2020-04-17 14:34:37,940	Generated vectors
399	2020-04-17 14:34:37,940	Completed caluculatesimilarity
400	2020-04-17 14:34:37,940	Starting caluculatesimilarity
401	2020-04-17 14:34:37,940	Generated vectors
402	2020-04-17 14:34:37,940	Completed caluculatesimilarity
403	2020-04-17 14:34:37,940	Starting caluculatesimilarity
404	2020-04-17 14:34:37,940	Generated vectors
405	2020-04-17 14:34:37,940	Completed caluculatesimilarity
406	2020-04-17 14:34:37,940	Completed find_cosine_similarity
407	2020-04-17 14:34:37,940	Completed syntactic_similarity_operation
408	2020-04-17 14:34:37,940	Starting display questions

Figure 8.6: Log File

This image is that of the log file which helps to identify the source of the error in case of fatal errors, each row contains a time stamp and information about the operation being performed.

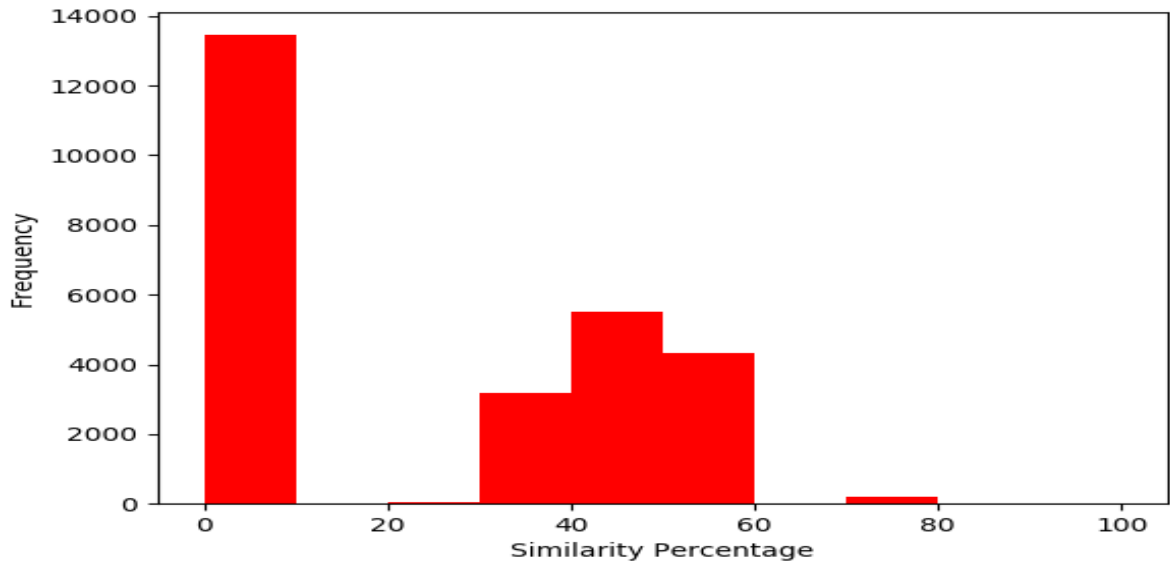


Figure 8.7: Similarity Graph

Above plot represents the frequency of questions falling in a similarity percentage bracket. This graph helps to understand that usually for a syntactic system majority of questions can be orthogonal or near orthogonal if there are multiple ways to state the same information.

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

Conclusion states the outcome of the project report presented on Suggesting relevant questions for a query using NLP techniques. This chapter also includes the future scope of the project helping any individual or team taking up the same topic in future.

9.1 Conclusion

From the results carried out it can be concluded that, syntactic similarity has good rejection ability for dissimilar question but fails to accept semantically similar questions, on the other hand semantic similarity has the ability to detect similar questions but does not effectively reject the dissimilar ones. A combined system that makes use of both the techniques produces better results as compared to using only one.

We have successfully presented a preliminary project report presented on Suggesting relevant questions for a query using NLP techniques, illustrating in detail the system requirement, system implementation plan, approaches used, algorithms used and the results obtained that can be referred as a guide for expanding the scope of the system in future.

9.2 Future Scope

The system can be extended to provide advanced functionalities given that the proposed requirements are fulfilled. The system capacity can be enhanced using advanced algorithms and techniques. The future scope of the system can be:

- Use of Quantum Computing for determining similarity
- Using Indic NLP in the System
- Creating dynamic graphics as per the question

9.3 Application

The system has wide range of applications and happens to be a suitable fit where humans may have to interact with a system using natural language. Some are mentioned below:

- Holistic learning system

Many a times students while learning a topic or subject for the first time, may not be able comprehend the material and when they ask questions on an online portal, similar questions to theirs will help in holistic development immensely.

- E commerce Websites

Today the importance of e commerce cannot be overstated, the main aim of such sites being revenue generation, if user id provided to questions similar to his/her it will lead to reduction in searching time and hence increase in profits.

- Company Training Portals

The system implemented can help any new joiner by predicting the problems or queries that may arise in future for an employee and providing a solution before hand proves to be very helpful.

APPENDIX A
FEASIBILITY STUDY

A.1 Problem Statement Feasibility

This section presents the system using a mathematical model and also discusses the feasibility of the problem. The system analyses if the model suggested belongs to the class of P, NPC or NPH.

The system can be represented as set of six tuple as follows:

$S = (I, D, F, P, I', D')$ where $I = \{x : \text{input query}\}$

$D = \{y \parallel y \in \text{Database}\}$

$I' = P(I)$

$D' = P(D)$

$P = I \rightarrow I' \ \& \ D \rightarrow D' (\text{PreprocessingFunction})$

$F = \{I, D\} \rightarrow R^+ (\text{Function that calculates the similarity score})$

The function F used to find similarity is of two types, namely syntactic and semantic, the Pre processing function contains these sub functions:

- Punctuation and Stop words Removal : $O(n)$
- Tokenization : $O(n)$
- Stemming : $O(k)$ [10]

The each of the pre processing functions runs $\forall x \in D$

$$\text{TimeComplexity}(\text{SyntacticSystem}) = n * (2n + n + n + K) = O(n^2) \quad (1.1)$$

The Complexity for a computing semantic similarity is $O(n * \log(V))$ [11] where n is the total size of database and V is the size of unique words list, the semantic system performs this function n times. Also, we need to account for the time required for removing stop words which is linear with respect to the size of data.

$$\text{TimeComplexity}(\text{SemanticSystem}) = n * (n * \log(V) + n) = O(n^2 * \log(V)) \quad (1.2)$$

The self learning lambda system performs the task of finding the similarity n times, hence the overall complexity can be represented as follows:

Overall Complexity (Self Learning System) is n times TC(Syntactic System) plus TC(Semantic System).

Time Complexity (Self Learning System) = $n * (TC(\text{Syntactic System}) + TC(\text{Semantic System}))$

$$TimeComplexity(SelfLearningSystem) = n * (n^2 + n^2 * \log(V)) = O(n^3 * \log(V)) \quad (1.3)$$

Clearly the complexity can be expressed as a polynomial function of the size of the input, hence the problem of finding similar questions using NLU lies in the class of Polynomial Time Problems.

Appendix B

Details of Papers

Appendix C

Plagiarism Report

Chapter 10

References

- [1] W. H. Gomaa and A. A. Fahmy, "A survey of text similarity approaches," *International Journal of Computer Applications*, vol. 68, no. 13, pp. 13–18, 2013.
- [2] G. Sidorov, H. Gómez-Adorno, I. Markov, D. Pinto, and N. Loya, "Computing text similarity using tree edit distance," in *2015 Annual Conference of the North American Fuzzy Information Processing Society (NAFIPS) held jointly with 2015 5th World Conference on Soft Computing (WConSC)*. IEEE, 2015, pp. 1–4.
- [3] R. Mihalcea, C. Corley, C. Strapparava *et al.*, "Corpus-based and knowledge-based measures of text semantic similarity," in *Aaai*, vol. 6, no. 2006, 2006, pp. 775–780.
- [4] A. Islam and D. Inkpen, "Semantic text similarity using corpus-based word similarity and string similarity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 2, no. 2, p. 10, 2008.
- [5] Y. Song and D. Roth, "Unsupervised sparse vector densification for short text similarity," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 1275–1280.
- [6] A. Kashyap, L. Han, R. Yus, J. Sleeman, T. Satyapanich, S. Gandhi, and T. Finin, "Robust semantic text similarity using lsa, machine learning, and linguistic resources," *Language Resources and Evaluation*, vol. 50, no. 1, pp. 125–161, 2016.
- [7] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [8] C. Schwarz, "lsemantica: A command for text similarity based on latent semantic analysis," *The Stata Journal*, vol. 19, no. 1, pp. 129–142, 2019.
- [9] A. M. Gliozzo, "Latent semantic analysis for application in a question answer system," Apr. 28 2015, uS Patent 9,020,810.
- [10] X. Wan, "A novel document similarity measure based on earth mover's distance," *Information Sciences*, vol. 177, no. 18, pp. 3718–3730, 2007.
- [11] StackOverFlow, *Word2Vec Time Complexity*, 2019 (accessed March 3, 2020). [Online]. Available: <https://stackoverflow.com/questions/54950481/word2vec-time-complexity>

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]